Web Software Services

State Framework for Developing Web Applications

Building an enterprise driven approach to software product development and hosting

An Executive Overview

Document Version 1.1.5

Prepared by: ITS/Web Software Services

Printed: Monday, January 13, 2003

Table of Contents

Table of Figures

1. Document Overview

This document is an executive overview of the ITS Framework for Developing Web Applications. Last year the Web Software Services Group recognized the need to work within a framework. There have been many discussions and brainstorming sessions since. Over the past year some steps have been taken towards the implementation of this framework's vision. More recently Douglas Lee and David Romney were given direction from Kerry Huntington to identify what the framework being practiced was and to fill in the holes.

This document is preceded by a definition document prepared by Douglas and David. It was presented to Kerry on October 18th, 2001. The definition document is a concise definition of the current framework and our expansion of it. It provides the initial direction for the framework to be fully specified from.

This document is one of four documents created to describe the framework. It is intended to provide a high level overview if our framework. This overview describes how the framework addresses the issues of the software product lifecycle.

In the addendum "The Process of Developing the Framework", you will find a brief description of the process we are following to identify, describe, and construct the framework. .

1.1. Purpose

The purpose of this document is to describe Web Software Services' software product lifecycle to external organizations.

1.2. Scope

The scope of this document is to provide no more than an executive level description of what the framework is.

2. Framework Overview

It is important for ITS to address key issues concerning the software product lifecycle in order to produce quality software products. The ITS Framework for Developing Web Applications is our strategy for addressing these issues and providing control over the software product lifecycle. This strategy does not represent organizational structure, but rather product development and delivery processes.

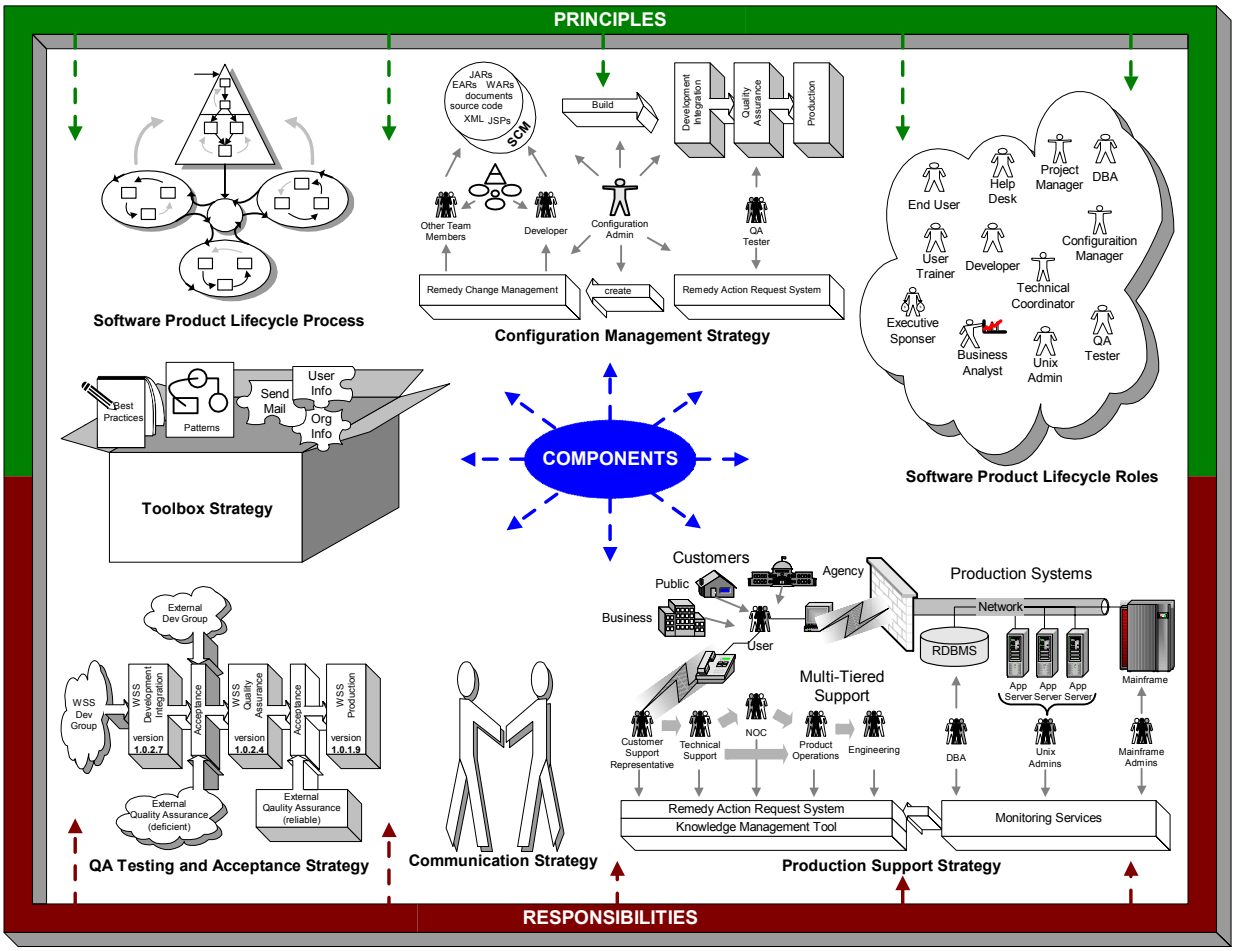

Figure 1: High level view of the framework

The framework is divided into the following areas:

- Principles
- Responsibilities
- Components

2.1. Principles

Figure 3 shows the principles as the top half of the framework's frame. The arrows pointing down from the top indicate that the principles are reflected in each framework component.

The principles give the team focus and direction as they perform their various roles. The table below describes each principle in more detail.

Principle	Description
Good communication	We focus on communication with external organizations as well as within ITS.
Staying customer driven	We focus on building the product the customer is asking for.
Keeping it simple	We work with the idea that the simplest solution is always the best solution.
Producing quality software	We produce quality by utilizing the components in the framework.

2.2. Responsibilities

Figure 3 shows the responsibilities as the bottom half of the framework's frame. The arrows pointing up from the bottom indicate that the responsibilities are crucial to the success of each component.

The software product lifecycle roles are responsible for practicing the framework's principles exercising his/her empowerment, documenting his/her work, regularly reviewing his/her work, and following the framework's process. The framework helps the individual fulfill his/her responsibilities. Roles and process are clearly defined, keeping things simple and easy to understand. The necessary tools and environments are provided to further simplify the work. The framework promotes a team approach while recognizing the talent and contribution of each individual.

2.3. Components

Figure 3 has seven separate diagrams representing each component of the framework. Component diagrams are shown to be contained within the framework.

The components section is where the framework's mechanics are described. They provide the tools and mechanisms necessary to organize and control the software product lifecycle.

The framework components consist of the software lifecycle roles, the software development process, the configuration management strategy, the toolbox strategy, the support strategy, and the communication strategy.

2.3.1. Software Product Lifecycle Roles

Figure 3 shows this component in the top right corner. The cloud is used to emphasize that lifecycle roles are independent of organizational structure.

The software product lifecycle roles define individual responsibilities, environments, interactions, and purpose. The definitions give context as to where in the software product lifecycle the role is needed and how the role is used.

2.3.2. Software Product Lifecycle Process

Figure 3 shows this component in the top left corner. The triangle represents the feasibility study phase and business study

phase. The three ovals represent the iteration phases. The small circle in the center represents the iteration planning transition phase.

The framework's software product lifecycle process incorporates the Dynamic Systems Development Method[1] (DSDM). We took current procedures and applied DSDM to fill in the process holes. The process remains iterative and incremental.

2.3.2.1. Iterative and Incremental Process

The iterative nature of the process allows us to control and accept changes in requirements. We understand that business objectives may change during the product development lifecycle.

The incremental nature allows stakeholders to see what is being built before the entire system is delivered. Initially, stakeholders review prototypes to help solidify requirements early in the process, to minimize feature changes later on. Additional stakeholder participation in the review of incremental deliveries keeps the development in line with the products vision.

Iteration planning allows developers to understand the chunks of work to be completed. While incremental deliveries demonstrate what is being built to stakeholders. A single incremental delivery includes one or more development iterations. This iterative and incremental process minimizes the possibility of requirements change due to communication deficiencies. It does not eliminate the possibility of legitimate change in business objectives, but may serve to clarify those that currently exist. Another key benefit is that Quality Assurance can do incremental testing as the system is being built. The alternative to incremental delivery and testing is the waterfall approach which delays the discovery of software defects and requirements problems until an entire system is delivered.

2.3.2.2. The Flawed Waterfall Approach

The framework's process is iterative and incremental as opposed to waterfall. The waterfall approach to software development is its predecessor. The waterfall approach has unnecessarily long phases where systems are over designed. After which they are built in their entirety. Once development is done the system is tested in its entirety. After testing the customer finally sees what he/she bought.

This approach has significant risk across the software product lifecycle. Issues discovered at the end of successive phases can cause the software product lifecycle to return to previous phases where resolutions are attempted. These retro fitted solutions can cause:

- Risk to not be mitigated early in the software lifecycle
- Architectures to be patched and become dysfunctional
- Business objectives to be compromised
- Entire projects to be scrapped and started over

In the fast paced world of software development today, organizations need to efficiently handle change in business requirements and risk discoveries. The waterfall approach is not an effective mechanism for dealing with this.

[1] To learn more about DSDM visit http://www.dsdm.org/index.asp

2.3.2.3. Software Product Lifecycle Phases

The component nature of the software product lifecycle yields provisions for other agencies to participate in any phase. The software product lifecycle phases are:

1. Feasibility study
2. Business study
3. Functional model iteration
4. System design and build iteration
5. Implementation
6. Production support

Product development starts with a feasibility phase. The deliverables of this phase are a Feasibility Study Report (with an initial cost estimate), and an Outline Plan.

After the delivery of the feasibility study report we begin the business study phase. In this phase the business rules and technology dependencies are identified and explored. Requirements are prioritized according to business risks, technology risks, and architectural importance. The deliverables of this phase are a Business Area Definition, an Outline Prototyping Plan, and the System Architecture Definition.

Now that the business study phase is complete we are ready to spin into a series of iterations, transitioning between the three iteration phases. The first of which is the Functional Model Iteration Phase. Artifacts produced during this phase are used in the Design and Build Iteration phase where the software product is constructed. The next phase is the Implementation phase where the software product is released and put into production. An important part of the implementation phase is the delivery of all product documentation. This includes user guides, training material, feature lists, and deficiency (bug) lists. This information is critical in helping the customer use the software product effectively.

Once the system is in production the software product lifecycle enters into the production support phase. Production support provides the customer with access to a help desk to find solutions for user problems. Production support also provides stakeholders with site monitoring services over the Internet. These monitoring services and user help desk calls feed into the Action Request System from Remedy. This system assures that production problems are captured so that they can be resolved through the appropriate change management process.

2.3.3. Configuration Management Strategy

Figure 3 shows this component at the top in the middle. The diagram shows how the configuration administrator is central to the execution and monitoring of the configuration management strategy. It shows the use of Remedy to handle trouble tickets and change management. It shows the migration path of software product from development integration to quality assurance to production.

The framework's Configuration Management (CM) strategy first deals with the management of source code and other project artifacts. The framework's source control system stores code and other project artifacts in central repositories. These repositories can be accessed securely from remote locations so that the team can work collectively.

The package and directory strategy provides consistency for team members as they move from

project to project. This allows team members to hit the ground running as new projects are taken on. The packaging strategy promotes the creation of software components (black boxes). This allows us to introduce new software components or replace old software components without redefining our package structure. The strategy is important. It prevents potential name collisions between organizations. A package naming convention is essential for establishing ITSs multi-agency hosting environment.

Procedures are in place to deal with changes to requirements, changes to architecture, and changes due to defect discoveries (bugs). We understand that business objectives change, and new risks are discovered as development moves forward. We have a change management strategy, which uses change requests, and change notifications to handle this.

Change management is also used to control the migration of the software product from the development environment to the quality assurance environment to the production environment. Part of our development process is to outline a release strategy, which shows these migrations for each version of the software.

We control software builds and releases through versioning to track what features and defects exist in which versions of the software. This helps the Web Software Services Group produce quality software with the features the customer is expecting. It also allows us to fix defects found in the production code base. This would be very difficult without the use of a source control system and a versioning strategy.

2.3.4. Quality Assurance Testing and Validation Strategy

Figure 3 shows this component in the bottom left corner. The diagram shows how quality assurance is the gateway to production from development. It also shows pathways for external organizations to use ITS quality assurance before migrating software product to production.

The Quality Assurance (QA) Strategy provides the mechanisms for a controlled migration of software from the Development Environment through the Quality Assurance Environment to the Production Environment. These QA controls from ITS are available for use by both internal ITS groups, and organizations external to ITS.

2.3.4.1. Internal Use of ITS QA

Internally, ITS QA controls provide the development effort with incremental testing of software components as they are built instead of months later. The nature of the iteration phases in the framework's Software Product Lifecycle Process make this possible. The benefit of building and testing components incrementally is that a more reliable software product is produced.

Internally, ITS QA controls qualify a software product as a candidate release ready for production based on quality of product and the level of compliance with business requirements. Software products already running in production need to be protected from other rogue software products. ITS QA acts as the gatekeeper, allowing only stable software products into production.

2.3.4.2. External Use of ITS QA

Externally, ITS QA controls provide software product testing for organizations, which have deficient or no QA process. Primarily these services are available for organizations which intend to host with ITS. ITS may optionally provide QA controls for organizations who have a non-ITS hosting strategy. These organizations will receive a lower priority when QA has resource constraints.

Externally, ITS QA controls qualify an external software product candidate release as production ready based on quality of product only. Organizations that intend to host with ITS must go through ITS QA control. As mentioned above ITS QA is the gatekeeper of production.

Our quality assurance group is a reliable source for qualifying software as production ready. In the future our hosting group might consider QA controls from external organizations to be reliable. For now the preferred route of acceptance is through our internal QA controls.

2.3.4.3. Incremental vs. Waterfall Testing

Through incremental builds and releases ITS avoids one of the pitfalls of the waterfall approach to software development. In the waterfall approach, large phases of a software product or even an entire software product would be built and then turned over to QA for testing.

It is likely that external organizations will deliver entire systems following the waterfall approach to software development. These organizations should be aware of the danger to their project timelines if QA rejects their software in its entirety. Re-scheduling with QA for testing will depend on resources and current workload. Since QA supports multiple development efforts, testing cannot be disrupted to accommodate re-testing of deficient systems.

2.3.5. Production Support Strategy

Figure 3 shows this component in the bottom right corner. This shows general representations of the three main focuses of the Production Support Strategy; Production Systems, Customers, and Help Desk. Relationships to Monitoring Services and the Remedy Action Request System are also described.

The framework's Production Support Strategy is implemented as the last phase of the Software Product Lifecycle Process. This phase occurs after a version of the software product has been released to production. There are three main areas of concern in this strategy. First is the monitoring and administering of the systems in the production environment. Second is providing the customer with access to the product and support for it. Third is the handling of trouble tickets as a result of customer issues.

The production environment is maintained and managed by Database Administrators, Unix Administrators, Mainframe Administrators, Security Administrators, Network Administrators, and Hardware Technicians. They are responsible for setting up and administering the systems in the production environment. These systems include databases, Unix operating systems, mainframe operating systems, legacy systems, application servers, portal products, networks, firewalls, and hardware configurations.

Customer access to the software product and product documentation, is provided through the use of the Internet. This provides a single access point for the software product and product information, which makes it easy to maintain as future versions of the software product are released into production.

The customer also needs the ability to interact with ITS when customer issues arise. ITS provides a multi-tiered support hierarchy:

- Tier One: Customer Support
- Tier Two: Technical Support
- Tier Three: Network / Server Operations Center (System Administrator)
- Tier Four: Product Operations Engineering
- Tier Five: Engineering

All of the tiers utilize the Remedy Action Request System for logging and escalating trouble tickets. A central knowledge management tool is also provided for finding solutions to past problems.

2.3.6. Toolbox Strategy

Figure 3 shows this component on the left side in the middle. This diagram demonstrates the idea of keeping common tools in one location for reusability.

The primary function of the Toolbox Strategy is to capture and reuse what is learned in the Software Product Lifecycle Process. The harvesting of this information is organized into reusable components and patterns. Tutorials and examples are then built to help developers use the toolbox components and patterns. Documents are also provided as part of the toolbox. These documents give developers common purpose and focus. The toolbox as a whole provides the material needed to foster good skills among the development staff.

2.3.6.1. Components and Patterns

During the development lifecycle reusable components and patterns are discovered. The discovery occurs in various ways. Developers may submit candidate components or patterns for consideration. Senior developers may be tasked with harvesting components and patterns from existing code bases. Industry standard components and patterns may also be included.

Toolbox components and patterns are rigorously tested and documented before they are made available for general use. This ensures that the toolbox does not waste developer time with debugging common code. The idea instead is to reduce development time by reusing well tested, plug and play solutions.

2.3.6.2. Tutorials and Examples

Addressing the education issues of developers using the toolbox is important. If the developer can't understand how to use a particular toolbox resource he/she won't use it. Or there will be one expert per toolbox resource who must explain the resource again and again to different developers.

One of the best ways to teach is by example. Before components are made available for general use, they must be well documented. The documentation will include tutorials and / or examples.

2.3.6.3. Documentation

The toolbox is the home for all documents that define and specify the development framework. This includes the document that you're reading now in addition to all other supporting framework documentation that provides a more granular view of the process.

Other development documents will be published in the toolbox such as white papers, articles, and other recommended reading materials.

2.3.7. Communication Strategy

Figure 3 shows this component at the bottom in the middle. This diagram is simple intentionally. It represents straightforward communication between people.

One of the framework's principles is the focus on good communication between individuals and between organizations. Communicating well throughout the Software Product Lifecycle reduces the risk of developing the wrong software product. Good communication also increases the efficiency of the Software Product Lifecycle and increases it's reliability.

Stakeholders participate early in the Software Product Lifecycle Process in order to communicate the software product's vision initially. Stakeholders participate later on by providing feedback during key demonstrations of the software product. This ensures that the software product is developed according to the stakeholders' vision, thereby minimizing breakdowns in communication.

Documentation produced during the Software Product Lifecycle Process is made available to everyone involved. This improves communication between team members by providing a common source of documentation that describes the software product.

The toolbox serves as a communication vehicle for sharing common components and patterns. This significantly reduces duplicated development effort.

The Iteration Plan breaks down the work into manageable pieces, which allows project managers to communicate effectively with developers and stakeholders. Project managers need to know what developers are working on and what work has been completed. This information allows project managers to communicate progress and status with stakeholders.

The Configuration Management Strategy improves developer collaboration by providing centralized repositories for code and documentation. This reduces the amount of trivial communication between team members in order to keep track of current code and documentation.

Change management procedures provide visibility to changes in the system. Change visibility ensures that the appropriate people handle the areas of the software product affected.

QA improves communication between development and production. It establishes a central,

repeatable process for qualifying a software product as production ready.

The Help Desk facilitates customer communication with ITS when issues arise. The call escalation procedures behind the Help Desk ensure that unresolved issues are communicated to the appropriate people.

3. Summary

The ITS framework provides structure and uses what was already in place and filled in the holes. The framework does not disrupt current process but instead enhances it. These enhancements will help the organization mature. There is an industry standard for measuring a software organization's maturity, the Capability Maturity Model (CMM).[2] The Software Engineering Institute (SEI) at Carnegie Mellon University created CMM. Our framework naturally increases our CMM rating.

[2] More information is available on CMM at www.sei.cmu.edu/cmm